NORTH-HOLLAND

The Functional Architecture and Interaction Model of a GENerator of Intelligent TutORing Applications

A. Kameas
Department of Computer Engineering and Informatics, University of Patras, Greece

P. Pintelas
Sector of Computational Mathematics and Informatics, Department of Mathematics, University of Patras, Greece

This article presents the functional architecture and the interaction model of GENITOR, a generator of Intelligent Tutoring Systems (ITSs). The major design issues that arise during the functional design and development of an ITS-generator concern the provision of mechanisms capable of supporting the description of the teaching subject domain and the specification of pedagogical features of the ITSs that it produces. The system described in the article provides ready-to-use yet flexible solutions for the instructional features of the developed ITSs. In this way, authors can concentrate on the compilation, description, and design of training material that relates directly to their area of expertise. Interaction between GENITOR and authors has been designed using IDFG, an interaction model that supports the representation of multiple aspects of interaction, including data flow, control flow, and task decomposition. As a consequence of using this model, the tasks that make up the authoring process are visualized by a number of authoring tools that are grouped into three subsystems: the reusability subsystem, the authoring subsystem, and the execution subsystem. The functionality of these subsystems and the tools they contain are described, as well as interaction with the overall system. Finally, conclusions drawn from the development of two ITSs using the system are presented in order to validate the design.

1. INTRODUCTION

The major design issue that developers of generators of Intelligent Tutoring Systems (ITSs) face is the provision of mechanisms capable of encoding the requirements of the different components of the tutoring process. Mechanisms of this kind have to support the representation of what information is to be taught (the domain) and of how the training process can be carried out as efficiently as possible (the instructional strategy). The former includes all kinds of domain knowledge (declarative, procedural, etc.). Instructional strategies heavily depend on the components of the training process: who will receive the information being taught (the trainee), in what context the teaching process has to take place (the tutoring discourse) and which means are used (the tutoring interaction) (Rickel, 1989; Mispelkamp, 1992).

Several design alternatives have been proposed, but no system has been implemented so far that meets all these design issues satisfactorily (Gerogiannis et al., 1993). Proposed solutions can be broadly classified into two groups, those that use AI techniques, and those that do not. Several commercially available authoring systems exist that exhibit impressive multimedia presentation capabilities but make limited provision for instructional design. On the other hand, there are systems developed in the context of research projects that use expert modules to meet the design requirements mentioned above.

One such system, GENITOR,[1] is presented in the article.[2] Design decisions were based on the results

Address correspondence to Prof. P. Pintelas, Department of Mathematics, University of Patras, Greece.

[1] Genitor (pronounced je'nitor) is a Greek word that means the one who gives birth, who generates.

[2] The focus of presentation will be on the design of the internal model of the authoring process that the system uses to interact with authors.

J. SYSTEMS SOFTWARE 1997; 36:233–245

655 Avenue of the Americas, New York, NY 10010

0164-1212/97/$17.00
SSDI 0164-1212(95)00099-M

of a survey of other authoring tools combined with a requirements analysis of a target user group (Pintelas and Kameas, 1993), that consisted of medium or high-level managers who needed to train their personnel in the use of methodologies of a certain kind.[3] Several interesting characteristics surfaced during analysis: most important of all, managers recognized the need to use computer-based tools for methodology training (Pintelas et al., 1992). They were, however, unwilling to invest time in developing a "polished" training application, but instead, they preferred to focus on describing the knowledge that is to be taught, using the simplest instructional techniques (which sometimes reduced even to a mere presentation of that knowledge). Consequently, they were not satisfied with the commercially available authoring tools, which mostly supported impressive presentations of unstructured knowledge.

The proposed system provides authors with tools that may be used to completely represent the knowledge domain of a training application, while ready-to-use solutions are provided for most of the authoring activities that relate to the specification of instructional strategy. It adopts an intermediate solution between authoring environments that include all necessary tools in their architecture (tools of this kind are described in Otsuki and Takeuchi (1985); Mispelkamp (1992); Wallsgrove (1992)) and those that provide a platform for integration of other existing tools (such as Derks and Bulthuis (1992); Philips and Nunn (1992)). Development of a training application using the former is usually pedagogically-driven, while the latter place emphasis mainly on the reusability of intermediate courseware products of different complexity levels (Olimpo et al., 1992).

The system user interface is "minimalistic" in the sense that only the necessary interaction elements are used, supported by a consistent interaction metaphor. Impressive features, such as natural language processing, or drag and drop facilities are not supported in an effort to make the system usage "transparent" even to authors who are noncomputer experts.

GENITOR produces intelligent training applications in subjects that are not necessarily related with each other, unlike systems like GUIDON (Clancey, 1987), SEDAF (Aiello and Micarelli, 1990), or SOPHIE (Brown et at., 1982), which generate and solve different problems in the same teaching subject domain. Applications developed with it are stand-alone in the sense that they may be used independently of the system.

The next section describes the authoring process, as it is represented in GENITOR. Section 3 contains a description of the system-author interaction, and Section 4 presents the model that was used for interaction specification. Section 5 presents the functional architecture of the system. A validation of system design and operation is contained in Section 6. In the last section, conclusions drawn and experience gained from system development are presented, together with future research directions.

2. SUPPORT OF THE AUTHORING PROCESS

Although no standard methodology exists for CAI systems or ITSs design and development, the proposed life cycle models (Keller, 1987; Roblyer, 1988; Alessi and Trollip, 1991) have many phases in common (Gustafson, 1991; Uden, 1992), including requirements analysis, courseware specification, design, implementation, and evaluation and revision.

GENITOR can be used for the last three phases of the life cycle of training applications. The initial activities of the development cycle that amount to material compilation and course planning have to be carried out manually. Before proceeding with actual development, authors should design the application knowledge domain base. A training application developed with GENITOR attempts to transfer two kinds of knowledge: procedural knowledge on how to apply a methodology, and declarative knowledge that provides a theoretical background for application of the methodology.

A *methodology* is any procedure that consists of distinct, partially ordered tasks, activities to carry out each task and results (outcomes) of each action called artifacts. Methodologies that can be taught with ITSs developed with the system have well-defined properties (Pintelas et al., 1992) and must be of a certain, albeit general enough, internal structure. Such an ITS aims at transferring to the trainees not only the correct ordering of carrying out tasks and activities, but also the ability to recognize the appropriate context of application of the methodology.

The declarative knowledge in a GENITOR application consists of Application Learning Units. A *Learning Unit* (*LU*) is an elementary block of knowledge (a piece of text, a picture, etc.) composed of a body with the LU content and a set of static descriptive attributes (Kameas and Pintelas, 1994). LUs can

[3] This analysis was carried out at first by using questionnaires, and, subsequently, with demonstrations of prototypes of the system.

J. SYSTEMS SOFTWARE
1997; 36:233–245

be used independently of any instructional strategy and can take part in constructs called *Application Learning Units* (*ALUs*), which form the hypermedia domain base of each application. The properties of ALUs are described by two classes of objects (Yazdani and Pollard, 1992): those that contain static information used to identify the unit and those representing dynamic information that describes the behavior of the unit.

An approach based on the Discourse Management Network (DMN) (Woolf, 1987) mechanism is used to model tutoring discourse. A DMN is a finite state machine that uses nodes and arcs to store information; arcs are activated depending on this information and external input. Conceptually, a DMN is a top-down refinement of high-level tutorial goals through the strategies and tactics that implement them (Rickel, 1989). The learning scenario embedded in an application developed with the system is called *learning cycle* and is made up of stages. A *stage* is an object that implements one pedagogic state of a DMN and constitutes an integral application module with respect to instruction.

GENITOR adopts a two-level structuring of the instructional strategy (Pintelas et al., 1992). Initially, authors have to define the learning cycle of the application by specifying which stages it includes. Since stages can be regarded as the pedagogic states of a DMN, at this level, authors compile the instructional design theory (Gagne et al., 1988) to be followed.

Then, authors have to specify the type and configuration of stages that will make up the learning cycle, out of seven predefined stage types that the system offers. Each stage encompasses an instructional strategy that is defined by a set of tutoring actions that the system will take when the stage is executed during training. Each action represents a tutoring objective; these actions implement the strategic states of a DMN. Authors can select which of these actions will be active during ITS execution.

Each tutoring action offers the trainees a set of operations that can be used in order to achieve the training objective that it represents. These are made available through the user interface of the ITS and depend on the tutoring context. The tactical states of a DMN correspond to the set of operations or functions that are offered to the trainees during tutoring.

This is an intermediate solution between guiding authors through the phases of some instructional development model, enabling them to develop (probably from scratch) a tutoring application, or providing lesson abstractions (templates) onto which authors will be building a "new" course. The former approach is more flexible but results in a lengthy development process, while the latter produces courses quickly, the functionality of which is unavoidably constrained. After all, such a system should be easy to use and at the same time make available the necessary instructional design expertise in an adaptive way (Duchastel. 1990; Spector and Muraida, 1992).

In order to overcome the complexity of the authoring process which is due to the highly dynamic nature of tutoring process, GENITOR employs two expert systems (Zaharakis et al., 1994). Methodology Expert System (MES) supports the description of the structure and dynamics of the methodology, during authoring and is responsible for its presentation during tutoring. An expert system was used for this task in order to relieve the authors from having to anticipate all the possible (correct and incorrect) sequences of combinations of methodology elements that trainees may form and then provide system responses for each. Authors simply describe these elements; it is the task of MES to ensure their valid ordering and to mediate tutoring interaction. Domain Expert System (DES) is employed during the presentation of ALUs. Again, the decision to employ an expert system aims at relieving the authors from having to explicitly describe all the associations between ALUs and the parts of the learning cycle.

During application design, ALUs (more specifically, the description of their behavior) may be added to it even if the constituent LUs (the ALU contents) have not yet been constructed. Integral parts of the methodology (i.e., a complete task) may be tested without the whole methodology being completed. Furthermore, authors may add the stages of the learning cycle, even if the tutoring actions or operations of these stages have not yet been fully specified.

The well-defined internal structure of training applications developed with GENITOR permits the prototyping of any application part of any degree of completion. Nonexisting (but required) application parts are temporarily added as having "null behavior" and are treated in a default manner; they are used as place-holders during prototyping. In this way, authors can quickly develop a prototype of an ITS, overcoming the production paradox (Carroll and Rosson, 1987).

Training applications developed with the system have all the primary and most of the secondary instructional characteristics proposed in Merill (1983). No authoring language is supported because the adopted hybrid scheme of the authoring process

makes the need for such a language obsolete. All that is required is a good understanding of the authoring process and of the nature of the applications that can be developed with this system.

3. WORKING WITH GENITOR

The authoring process is regarded as a set of authoring actions, which are grouped with respect to their outcome. Each group corresponds to an integral authoring phase and is visualized with an *authoring tool*. All tools are accessible from within any tool, but authoring contexts may not overlap, allowing only one tool to occupy the screen at any time. Each tool is used for the development of the corresponding *tool object*, which represents a subgoal of the application construction process. The abstract notion of tool object physically corresponds to one application file. All tools will have to be used for the achievement of the ultimate goal: the development of a training application.

A training application developed with GENITOR consists of a set of information files, which, except LU content files, are made up of entries. A file entry is defined as a set of components; an entry component is an atomic application parameter. The development of the application is thus a repetitive process of file entry manipulation that can be decomposed into atomic subgoals of assigning values to entry components.

In order to overcome the assimilation paradox (Carroll and Rosson, 1987) and avoid any transference phenomena caused by the transfer of interaction experience with other systems, a uniform interaction metaphor is used for all tools. Tool user interfaces are consistent, look alike, and function more or less in the same way[4] by using interaction elements (such as menus, buttons, controls, etc.) that follow established standards.

The screen window of an authoring tool is divided into two areas: the information area at the left and the action area at the right half of the screen (Figures 1 and 2). The former displays the current state of the tool object in the main data area and the planned state transition in the current data area. Transitions on the state of an object are caused on a per entry basis: authors build the new entry inside the current data area, and then copy it into the main data area.

Inside the action area, the authoring actions that will cause a state transition towards the achievement of the subgoal represented by the tool are visualized. The authoring actions that assign a value to an entry component are visualized with *controls*. Two kinds of controls exist: *input controls* and *select controls*. When using the former, authors have to type the value of the component, while the latter present a list of predefined and mutually exclusive values for them to choose from. One input control (namely, "Add

[4] Only the various LU content editors do not conform to this model of interaction.

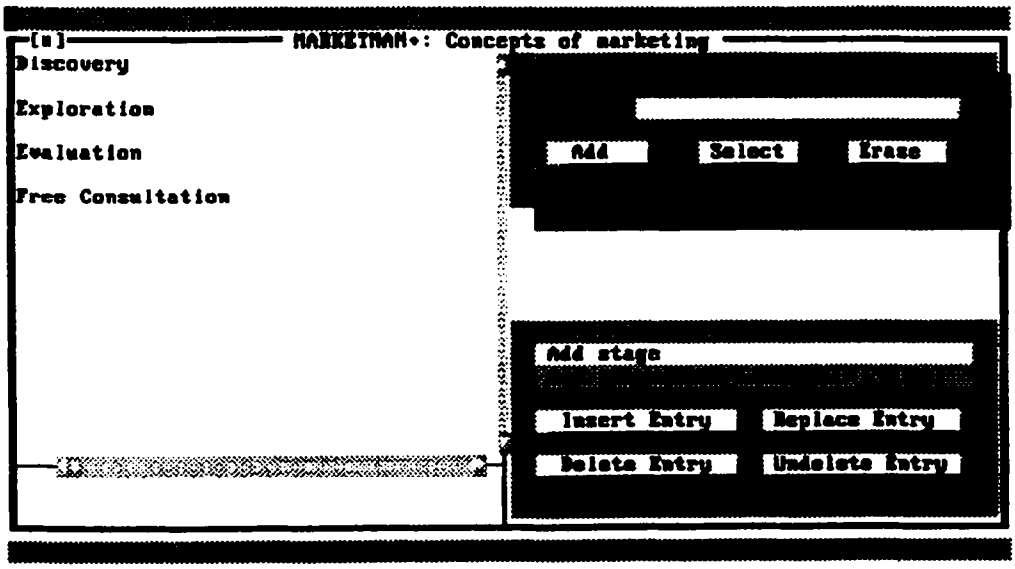

Figure 1. The user interface of the Discourse Manager authoring tool.

J. SYSTEMS SOFTWARE
1997; 36:233–245

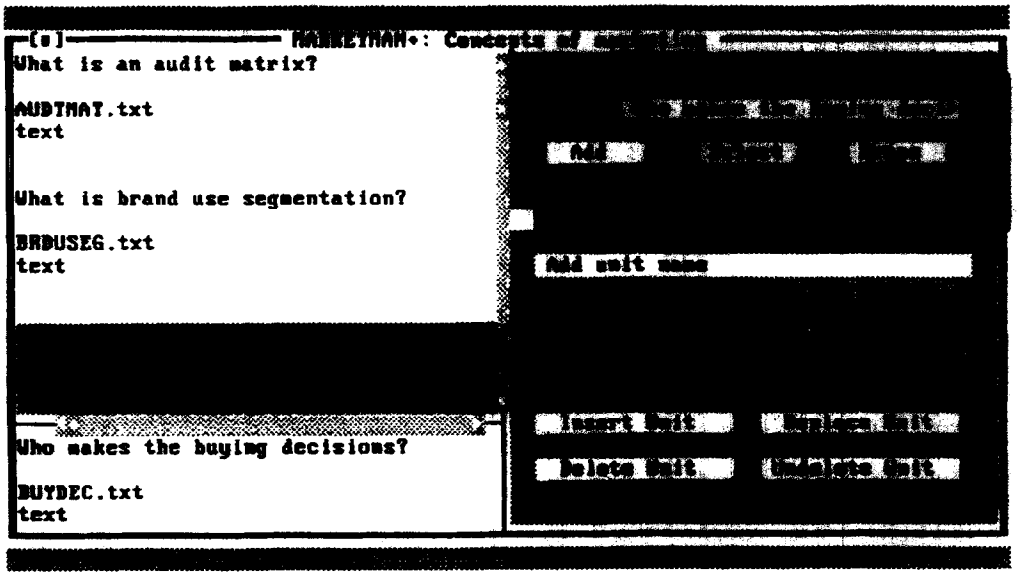

Figure 2. The user interface of the Domain Manager authoring tool.

Learning Unit"), is depicted in Figure 3. Using it, the authors can add an ALU to the application domain base by specifying its title. The title of the ALU must either be typed inside the blank area next to the word "unit", or be selected from a list of available ALUs (by pressing button "Select"). Pressing the "Add" button causes the addition of the ALU to the domain base, and pressing the "Erase" button causes the contents of the control to be erased.

Using all the controls of a tool, authors can specify an entire entry. The grouping of controls depends on the context modeled by the associated tool. For example, the tool responsible for the specification of the learning cycle of the application (namely Discourse Manager shown in Figure 1), contains input controls "Add Stage" and "Add Parameter List". The former is used for stage selection, while the latter is used for the description of stage execution parameters (i.e., termination criteria, etc.). Only one control may be open at any time, thus avoiding the overlapping of elementary authoring contexts.

Figure 3. One of the controls that represent elementary authoring actions.

Reusability is supported through the various *selectors*, which are a special kind of controls. Selectors are directly managed by the reusability manager of GENITOR. In effect, they provide the results of system-specified queries on the reusability bases of the system. These queries recall a list of application parts that have similar properties with respect to the purpose of the context from within which they were posed. This means that the contents of the list depend on the tool that was open and on the control that was used to activate the selector. Two kind of selectors exist: *single selectors*, which permit the insertion of only one object at a time, and *list selectors*, which are used to specify a list of objects for insertion.

In this way, authors may develop an application in a top-down way (first by designing its learning cycle, then the structure of its domain, and then by constructing the LUs, which have to be inserted from the reusability bases using the reusability manager), or in bottom-up (starting from the development of the LUs and moving towards definition of tutoring components), or in a mixed procedure. Selectors function as the channels that link together the phases of design and of implementation.

Activities that do not belong clearly in the authoring process, such as file loading, saving, communicating with other tools, or changing the context, are represented with *menus* or *buttons*.

Apart from what authors see, each tool tacitly supports the authoring operations and prevents them from making serious mistakes by holding internally a

representation of its operation, where *action closures* and *integrity rules* are represented. Only correct data are copied from a control to the current data area, which means that only correct elementary authoring actions are allowed to have any effect. Each tool performs a second round of data integrity checks when the contents of the current data area are to be copied inside the main data area, in order to ensure that a safe change in the state of the tool object is about to happen. Unacceptable data are not rejected but are marked as deleted, which means that it will not be taken into account during application execution. Actions that produce deleted data are considered as passive (Thimbleby, 1990) and have no effect on object state.

On the other hand, authors are never allowed to exit a tool in the middle of a sequence of actions, abandoning an incomplete plan, and leaving a subgoal partially attained. In such cases, each tool communicates with the author through *floating controls* and *messages*. Floating controls are used when the tool urgently needs information of a certain kind; they are two-way communication channels. Messages are used to inform the authors of a situation that has emerged. Each message clearly describes the current context, the action that led to the current state and the state itself and proposes the action to be taken; authors may accept or reject this proposition (sometimes there is no alternative but to accept the proposed system action).

4. THE INTERACTION MODEL

During the design of GENITOR, the IDFG model (Interactive Data Flow Graph) (Kameas et al., 1993) has been used to describe the interactive behavior of the system. This model regards interaction design as software process design and uses a high-level Petri Net model (Reisig, 1992) extended to include cognitive aspects of interaction (Kameas et al., 1994). IDFG models an interactive application as a set of communicating graphs; each IDFG is a bipartite graph.

Nodes are of two different types: *links* and *actors*. Actors represent the goals of different levels that authors may achieve by using the system. Links are used to describe events that take place in the system, actions that lead to the achievement of goals, the context inside which these actions may take place, or conditions that represent availability of actions or context. In addition, they represent roles in the authoring process and support data modeling. Links store *tokens*, which are transported across the directed arcs that connect actors to links and links to actors.

Actors can represent the entire goal-subgoal decomposition structure of user plans. Achievement of a goal causes the *firing* of the corresponding actor. Three kinds of actors are used:

- *action actors*, which model the low-level goals that can be achieved by using the interactive application.
- *context actors*, which represent the structuring of low-level goals into higher-level macro-goals which can be achieved through the user interface of the application. Context actors may be refined into structures of lower level actors.
- *library actors*, which are system-defined actors with predefined functionality. They are used during the refinement of context actors.

In this way, the implementation of high-level interface functions is separated from that of low-level authoring actions, thus achieving transparency in system usage and interface independence at the same time (Thimbleby, 1990). The number of action actors is finite and equal to all the commands supported by GENITOR. The task of context actors is to correctly interpret authors' actions in order to appropriately decompose their goals into subgoals and so that eventually the correct action actor will fire.

Links are typed; each link may store tokens of its type only, while the existence of a token has different meaning depending on the link type. Available link types are *user action*, *system action*, *condition*, *data*, *goal*, *incommunication*, *outcommunication*. This set of types permits the description of several perspectives of an interactive application (Kameas et al., 1994) (i.e., causation, cognitive aspects, system behavior, etc.), which, when combined, produce an integrated, consistent, and complete model of the interaction process (Curtis et al., 1992).

Execution (*firing*) of an actor is determined by rules associated with it; rules include the input links of the actor in their left side (if-part) and its output links in their right side (then-part). Then, tokens are consumed from those input links, and tokens are produced in those output links of the actor that take part in the rule that caused the firing.

At any moment, there exists a number of actors (the actor-ready list) that represent the goals available to the authors. Each goal can be achieved by the authors causing the appropriate event. In IDFG, a state is defined by the set of actors in the actor-

J. SYSTEMS SOFTWARE
1997; 36:233-245

ready list, or equivalently, the set of user or system actions that can fire the actors in the actor-ready list. Since these actions correspond to goals in a lower-level, it can be equivalently said that a state is represented with the set of goals that may be achieved as a consequence of user or system actions permitted by the actors in the actor-ready list. Each actor firing modifies the distribution of tokens on links and consequently produces a new state. Thus, commands are modeled with actor firings and sequences of commands with actor firing sequences. Every allowed author action must belong to a goal-leading sequence.

In Figures 4 and 5, two snapshots of the IDFG that represents interaction with the control of Figure 3 are depicted. Action actors are represented with black rectangles, context actors with bold rectangles, event links with bold circles, data links with diamonds, context links with dotted circles and condition links with plain circles. Black dots inside links are tokens.

All the actions supported by a control belong to the same context, as represented with goal links and are modeled with IDFG actors.

In Figure 4, context actor $A4$ represents the subgoal "Add Learning Unit" (link g). This actor belongs to context "Specification of the ALU library", as described by input context link c. Its firing rule is $c, g \rightarrow cOK, d1$. Library actors ANDS (stands for AND Start) and ANDE (stands for AND End) describe the internal structure of $A4$: in order to add a learning unit, authors must first "Specify Unit Name" (context actor $A5$) *and then* "Confirm Name" (action actor $A6$). Precedence among these subgoals is described with condition link *con*1 ("UnitName-Changed"). The specification of the unit name can be done in one of three ways: by typing directly inside the input line of the control (action actor $A3$), by using a selector (context actor $A1$), or by simply erasing the contents of the input line (action actor $A3$). The structure described by library actors ORS (stands for OR Start) and ORE (stands for OR End) means that it is enough for one of the actors to fire in order for the subgoal represented by context actor $A5$ to be achieved. Authors have to execute the appropriate actions in order to cause actor firing (i.e., the firing rule of actor $A1$ is: $c.A4.A5$, *PressSelectButton* $\rightarrow$ $c.A4.A50K$, *con*2).

By clicking on the "Add Learning Unit" item of the control menu, a token is produced in link g and

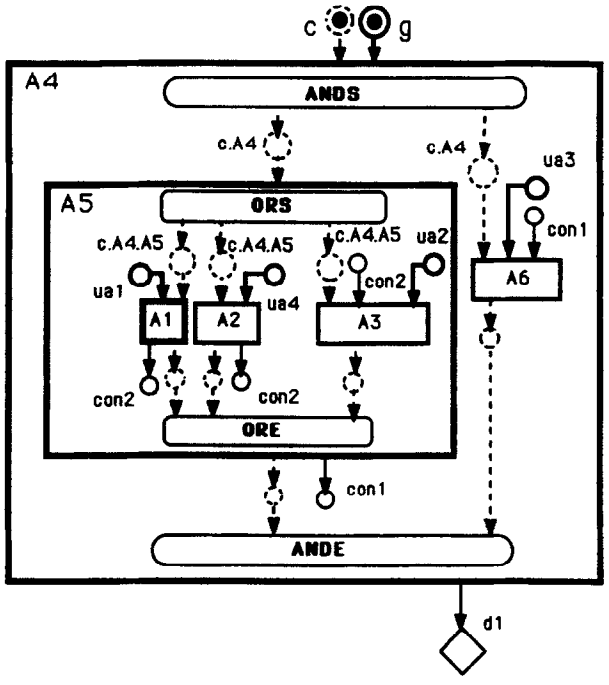

Figure 4. The IDFG that represents interaction with the control of Figure 3 (snapshot 1).

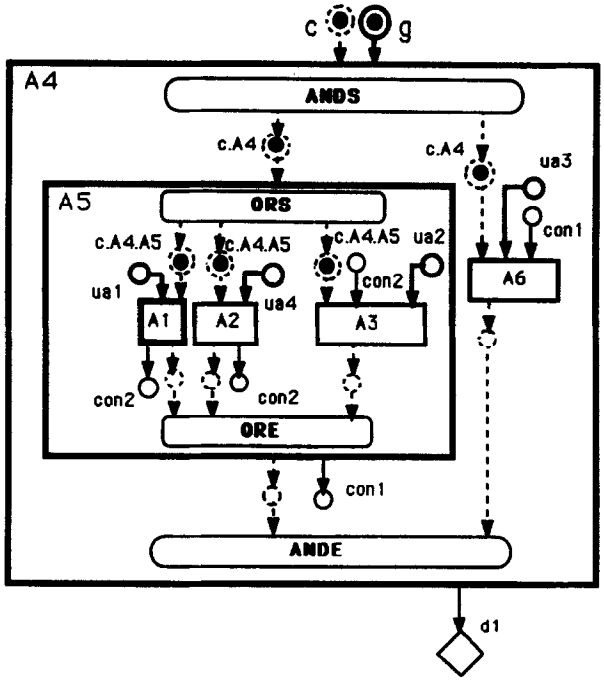

Figure 5. The IDFG that represents interaction with the control of Figure 3 (snapshot 2).

actor $A4$ fires (Figure 4). Then tokens are produced in links $c.A4$ and $c.A4.A5$, and actors $A1$, $A2$, and probably $A3$ and $A6$ (it depends on the value of links $con2$ and $con1$, respectively) are added to the actor-ready list (Figure 5). By clicking on the ADD button of the control (which is available only if the unit name has changed, as described by condition link $con1$), execution of $A4$ terminates and the data item "unit name" is returned in data link $d1$.

A correspondence exists between the structuring of interaction as described by IDFG and the user interface elements used. Tools are the highest level context actors, which are composed of context actors that represent controls, buttons, selectors, and menu items. These are eventually composed of action actors. Data links are used to transfer data to and from files and between the screen areas, while communication links are used for tool synchronization. Condition links are used for representation of screen situation. Links also perform integrity checking; an integrity error causes a system action that appears in the form of a floating control or a message. All these actions are explicitly represented in the user interface of the system, conforming to the principle of observability (Thimbleby, 1990).

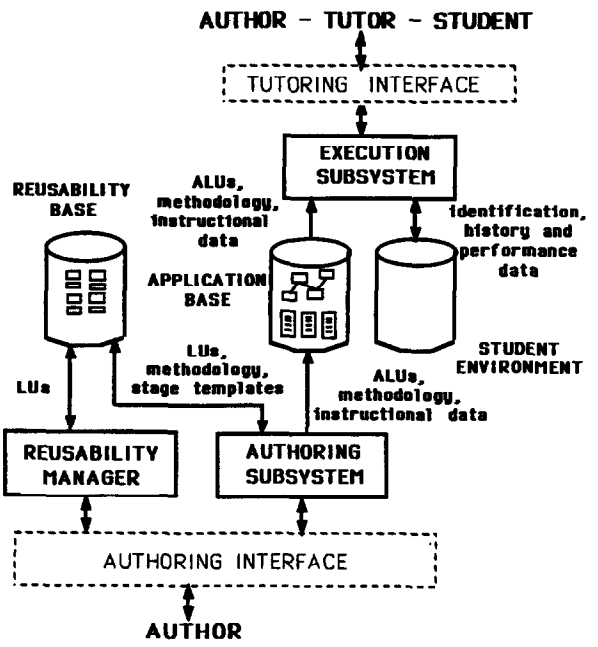

Figure 6. The functional architecture of GENITOR.

5. FUNCTIONAL ARCHITECTURE

GENITOR consists of three major subsystems: the *reusability manager*, the *authoring subsystem* and the *execution subsystem* (Figure 6). The first is used for storage, maintenance, and retrieval of application parts. The authoring subsystem is used for application development, while the execution subsystem provides a prototyping facility to authors and facilitates the integration and packaging of the application.

To compile declarative knowledge, authors must use the LUs of the reusability base and develop the hypermedia ALUs by assigning attributes to them that describe their role within the application. This is a time-consuming process which becomes even lengthier if no suitable monomedia LUs exist in the reusability base and have to be developed from scratch. The development of the procedural knowledge base requires different authoring skills, and its duration depends on the expertise of the authors with respect to the design of the methodology elements. The shortest authoring phase is the definition of the pedagogical aspects of the application, which include the definition of the learning cycle by selecting the stages and the stage types that will be included in the application, together with the special characteristics (pedagogic tasks and training operations) of each.

5.1 The Reusability Manager

Authors can develop learning material that is not pertinent to any application and store it in the reusability base for subsequent use. For the time being, LUs, stage templates, and methodology templates can be archived; the concept will be extended in the future to support all kinds of learning material that may be used in an application developed with GENITOR.

This subsystem includes a set of learning unit editors that may be used for the construction of monomedia LUs. Currently, LUs of types text, picture, and test are supported. Using the database archiver, authors may archive each LU. Executable programs are regarded as LUs of type external. The body of an LU is automatically copied to the appropriate directory, and the system produces an internal identification code to uniquely identify each unit. System designers have already included in the reusability base one template for the type of methodology that can currently be used and seven stage templates, one for each available stage type. With the tools of this subsystem, only the content and the objective attributes of application parts can be described; behavioral attributes and associations

can only be described in the context of an application, by using tools of the authoring system.

5.2 The Authoring Subsystem

The authoring subsystem of GENITOR includes a set of highly interactive tools that are controlled by the authoring subsystem manager, which constantly monitors interaction and is responsible for the overall control of authoring actions. The application configuration manager is used to describe general application features, such as title, author, version, etc., as well as the intended configuration of the application.

Each application includes its own domain base, which contains both procedural and declarative knowledge. The methodology manager is the authoring interface of MES. Using it, authors can perform an elementary knowledge engineering process, in order to describe:

- the structure (static contents) of the methodology, by retrieving the methodology template from the reusability manager and then specifying all the levels of methodology tasks and subtasks, the activities that make up each of them, and the artifacts that are produced as a result of the execution of each activity. Based on this description, MES can produce at authoring time the inheritance relations among the various elements of the methodology.
- the special vocabulary used by the methodology at-hand (that is, the appropriate terms that will be used in place of the general-purpose and system-provided "methodology", "activities", "artifacts", etc.). These will be appropriately displayed by MES during ITS execution.
- the ordering of the tasks and activities (dynamic behavior of the methodology). Instead of specifying some explicit order, authors need only describe a prerequisite relation among artifacts, by specifying which attributes must have already been produced in order to be possible for a new artifact to be produced. Based on this description, MES will be able to order the methodology tasks and activities at run time and decide which one should be carried out next.

The declarative knowledge of the application is constructed by combining LUs. Authors have to retrieve such monomedia units from the reusability base, relate them to the application context and combine them into hypermedia units (ALUs) by forming objects that act as hypermedia unit headers. These objects include attributes that are handled/evaluated by corresponding methods; authors select which attributes are to be associated with an object out of a set of available attributes with system-defined behavior. Using the domain manager, the application domain base can be constructed by including author-defined ALUs consisting of hypermedia headers and of the associated monomedia units. For the sake of completeness and for prototyping purposes, the system provides units with "null behavior", which are executed in place of missing or damaged units.

GENITOR provides a set of tools that may be used to define the learning cycle of the application. Authors must use the discourse manager to produce the application script file, where the learning cycle of the application is described, together with the prerequisites and termination criteria of each stage in the cycle. Then, for each stage of the learning cycle, authors have to retrieve the corresponding stage template from the reusability base, and then use the instructional manager to specify the application-dependent attributes of the stage, like title and configuration, the pedagogic functions (strategic states) that will be carried out by the stage, and the operations that will be offered to the trainees during execution of the stage (tactical states).

5.3 The Execution Subsystem

GENITOR supports the building of a library of training applications by using the same runtime support system for each of them. A result of this concept is the application menu that is presented to the application users. Before execution of an application can commence, the users' names and intended usage mode must be declared.

Each trainee can follow an individual learning trajectory, always within the limits set by the author, by selecting the next stage to be executed. After execution of a stage terminates, the system presents the trainees with a list of all the stages that have been executed, having the next stage in the sequence highlighted. The trainees can accept the proposal of the system, make a different selection, or restart the application.

The part of MES that is included with every application is responsible for teaching the procedural knowledge of the application and presents the trainees with a simulation of the methodology to be taught. The dialog is based on a constrained version of problem solving which engages the trainees in a game-like simulation (Burton and Brown, 1982): the problem posed each time is to find the correct

activity that must be taken next in order to advance inside the simulation of the methodology evolution. The choice must be made among the set of all activities that make up the methodology. Correct choices are awarded with GREs (GRades of Excellence). Depending on the stage type, MES operates in three modes: guiding, coaching, evaluating.

In the first mode, the role of MES is to simply present the procedural knowledge to the trainees. To this end, MES conducts itself a simulation of the methodology evolution by selecting each time the correct activity. The progress within the methodology is visualized to the trainees using animated charts and diagrams. In coaching mode, MES supports a learn-by-discovery process of the methodology. It is the trainees who must now select the next activity from the set of all methodology activities. The role of MES is to coach them (Nawrocki, 1987) by commenting on their selections, responding to their commands, and assuming control when the trainees appear to be lost. Finally, during evaluating mode, MES leaves control of the simulation entirely at trainees' hands and simply judges their selections.

DES is responsible for presenting the ALUs to the trainees based on the requirements of the instructional strategy because these are encoded in the strategic states of the stages and on the attributes of the ALUs, after evaluating them.

Each stage manages its own environment and determines its state of completion. The user interface of any stage contains at most three areas: procedural, declarative, and utility (Figure 7). The procedural area mediates interaction between MES and the trainees (it is the large white rectangle area that contains small boxes in Figure 7). It conveys their choices to MES and presents its response. The declarative area is used by DES to display the application ALUs at the trainees' choice (the bottom rectangle area in Figure 7, which contains titles of ALUs). One of these two areas may be missing from the interface of some stage types. The utility area represents all the tutoring operations and functions that are available to the trainees, as these are specified by the tactical state objects of the stage (the top line of dark rectangles in Figure 7).

Every application automatically constructs a student environment for each trainee. Within this environment, the application records trainees' identification, performance, progress and statistical data. In addition, the application maintains an elementary student model that classifies the trainees according to their overall performance and their latest response. This model is used by both DES and MES to reflect the knowledge levels and the misconceptions of the trainees. Collected performance data are made available for evaluation to tutors through the administrator environment, which is accessible by using an author-defined password.

6. VALIDATION OF DESIGN

GENITOR can be used by authors as a stand-alone development environment. Developed applications can be used for self-education, for group training, or even as "on-the-spot consultants". In order to validate an earlier version of GENITOR (called

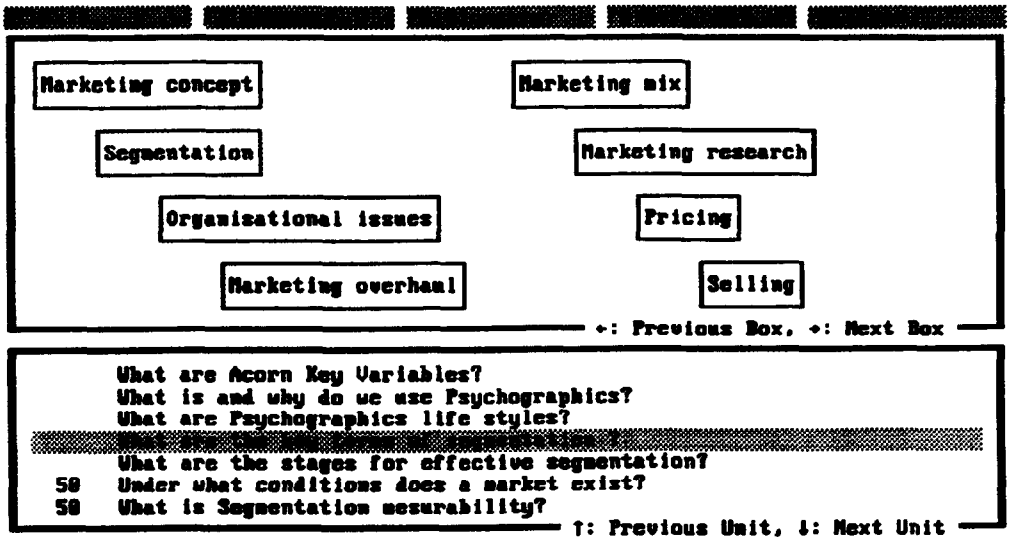

Figure 7. The user interface of an application stage.

GEPRIAM), two applications have been designed and developed (Pintelas et al., 1992; Kameas and Pintelas, 1994): METHODMAN II+ and MARKETMAN. The former is a four-hour self training application that aims at teaching the MEDOC methodology for software project management. METHODMAN II+ addresses the needs of engineers and technicians participating in innovative projects, as well as project managers who administrate such projects. The latter is an ITS for training in the use of marketing strategies which is addressed towards marketing executives or salespeople.

METHODMAN II+ was developed by a three-person team, which included members of the GENITOR design team. It uses (with slight adaptations) the ALUs and learning cycle of METHODMAN I, an ITS that was initially developed to demonstrate the need for training applications of this kind. As a result, development and validation of METHODMAN II+ took less than one personmonth, while METHODMAN I took over two personyears to develop (without the use of a generator, though).

MARKETMAN was designed and developed in about three personmonths by a three-person team which included an expert in marketing and two computer experts. It took a short course on the functionality of GENITOR to make them familiar with the system. The application uses the same learning cycle with METHODMAN II+ but an entirely disjoint set of LUs and ALUs.

The speedup in development time that results from the use of the reusability base depends on which part of the application is totally or partially reused. The design, development, verification, and delivery of an entirely new intelligent tutoring application of METHODMAN II+ size should take at most six personmonths.[5]

The development of those two applications established the capability of GENITOR to produce intelligent training applications on diverse domains having in common only the requirement that there exists a structured representation of the methodology to be taught (Pintelas et al., 1992); development was greatly facilitated by the inclusion of a methodology template in the reusability base. Furthermore, the provision of stage templates relieved much of the burden from authors during the specification of the instructional features of the application. However, GENITOR tends to be overprotective, performing too many data and application integrity tests.

[5] This effort should not be compared with that required by current authoring/presentation systems; GENITOR is an ITS-Generator.

7. CONCLUSIONS

In this article, the interaction model and functionality of GENITOR, an ITS-generator that addresses the needs of noncomputer expert authors were presented. GENITOR can be used as a stand-alone development environment by authors who need to encode their domain expertise in an application as quickly and efficiently as possible, without being much concerned with instructional issues.

The system was designed using object-oriented analysis and design techniques, while the IDFG model was used for the design of the user interface. The authoring subsystem was developed using Borland Turbo Pascal v 7.0 under MS-DOS v 6.2 in a PC-486 platform.

The separation of the development process from the application execution process, together with the adoption of object-oriented design techniques permitted the incremental system development, led to an open-ended architecture, supported the notion of an "application library", and will provide an easy maintenance/upgrade process. In this way, the problem of upgrading old applications can be easily solved, and the capabilities of the applications can be improved over time to meet emerging technology standards. The user interface of the system was developed with the TurboVision environment included in Turbo Pascal. The execution subsystem was developed under Turbo Pascal as well. In this case, object-oriented techniques were not adopted because this part of the system is rather computation than data-intensive. The authoring part of the two expert systems was developed using TurboVision; knowledge representation is frame-based, and their execution part was developed under Turbo Prolog v 2.0.

Currently, an effort is under way to have the system run under Microsoft Windows. In the next version of the system, several modes of operation (e.g., novice, expert, etc.) will be provided, with different grouping of user actions for each. Other improvements considered by the design team are the generalization of the functionality of the reusability base, so that application parts of any complexity and nature could be archived and the provision of a fully-functioning student model. All these extensions must impose the least possible cognitive load on the authors that use GENITOR; in any case, this was the driving guideline of the current design, as well.

ACKNOWLEDGMENTS

The first prototype of GENITOR was developed under the name of GEPRIAM within project GESEM of the EU program COMETT II (contract no 90/3/5081/Cb). The authors would like to thank the project teams from SYSECA SA, Ecole Nationale Superieure des Techniques Industrielles et des Mines d Ales, and Business School of the John Moores University of Liverpool for their efforts and contributions.

REFERENCES

Aiello, L., and Micarelli, A., SEDAF: An Intelligent Educational System for Mathematics, *Applied Artificial Intelligence*, 4(1), 15–37 (1990).

Alessi, S., and Trollip, S., *Computer-Based Instruction: Methods and Development*, Englewood Cliffs, Prentice Hall, 1991.

Brown, J. S., Burton, R. R., and Clancey, W. J., Pedagogical, natural language and knowledge engineering techniques in Sophie I, II, and III, in *Intelligent Tutoring Systems*, (D. Sleeman and J. S. Brown, eds.), Academic Press, 1982.

Burton, R. R., and Brown, J. S., An investigation of computer couching for informal learning activities, in *Intelligent Tutoring Systems*, (D. Sleeman and J. S. Brown, eds.), Academic Press, 1982.

Carroll, J. M., and Rosson, M. B., Paradox of the active user, in *Interfacing Thought*, (J. M. Caroll, ed.), MIT Press, 1987.

Clancey, W. J., Methodology for building an Intelligent Tutoring System, in *Artificial Intelligence and Instruction*, (G. P. Kearsley, ed.), Addison-Wesley, 1987.

Curtis, B., Kellner, M. I., and Over, J., Process Modeling, *Communications of the ACM*, 35(9), 75–90 (1992).

Derks, M., and Bulthuis, W., A framework for authoring tool integration, in *Learning Technology in the European Communities*, (S. A. Cerri and J. Whiting, eds.), Kluwer Academic Publishers, pp. 549–561, 1992.

Duchastel, P. C., Cognitive Designs for Instructional Design, *Instructional Science*, 19(6), 437–444 (1990).

Gagne, R. M., Briggs, L. J., and Wager, W. W., *Principles of Instructional Design*, Holt, Rinehart and Winston, Chicago, 1988.

Gerogiannis, V., Giakovis, D., Pintelas, P., and Kameas, A., Intelligent systems for education: an overview. Technical Report TR 93.10.21, Department of Mathematics, University of Patras, Patras 26500, Greece, 1993.

Gustafson, K. L., Survey of Instructional Development Models, ERIC Clearinghouse of Information Resources, Syracuse University, 1991.

Kameas, A., Papadimitriou, S., Pintelas, P., and Pavlides, G., IDFG: An Interactive Applications Specification Model with Phenomenological Properties, in *Proceedings of the EUROMICRO 93 Conference: Open System Design*, Barcelona, Spain, September 6–9, 1993.

Kameas, A., Gerogiannis, V., Diplas, K., and Pintelas, P., Encapsulating Multiple Perspectives in Interaction Specification, in *Proceedings of the EUROMICRO 94 Conference: System Architecture and Integration*, Liverpool, United Kingdom, September 5–8, 1994.

Kameas, A., and Pintelas, P., GENITOR: a GENerator of Intelligent TutORing application. Technical Report TR 94-01, Division of Computational Mathematics & Informatics, Department of Mathematics, University of Patras, Greece, 1994.

Keller, A., *When Machines Teach: Designing Computer Courseware*, Harper & Row Publishing, 1987.

Merill, M. D., Component Display Theory, in *Instructional Design Theories and Models: An Overview of their Current Status*, (C. M. Reigeluth, ed.), Hillsdale, Lawrence Erlbaum, pp. 279–333, 1983.

Mispelkamp, H., Generic tools for courseware authoring, in *Learning Technology in the European Communities*, (S. A. Cerri and J. Whiting, eds.), Kluwer Academic Publishers, pp. 585–593, 1992.

Nawrocki, L. H., Artificial Intelligence applications to maintenance training, in *Artificial Intelligence and Instruction*, (G. P. Kearsley, ed.), Addison-Wesley, 1987.

Olimpo, G., Choiccariello, A., Tavella, M., and Trentin, G., On the concept of reusability in educational design, in *Learning Technology in the European Communities*, (S. A. Cerri and J. Whiting, eds.), Kluwer Academic Publishers, pp. 535–549, 1992.

Otsuki, S., and Takeuchi, A., Intelligent CAL system based on teaching and learner model, in *Computers in Education*, (K. Duncan and D. Harris, eds.), Elsevier, 1985.

Philips, P., and Nunn, M., The relationship between PETE and PCTE, in *Learning Technology in the European Communities*, (Cerri and Whiting, eds.), Kluwer Academic Publishers, pp. 563–571, 1992.

Pintelas, P., Kameas, A., and Crampes, M., Computer-based Tools for Methodology Teaching, in *Proceedings of the 34th International ADCIS Conference: Empowering People Through Technology*, Norfolk, VA, November 8–11, 1992, pp. 341–355.

Pintelas, P., and Kameas, A., Experience From Using Information Technology in the Training of Managers, *European Journal of Information Systems*, 2(2), pp. 129–137 (1993).

Reisig, W., *A Primer in Petri Net Design*, Springer Berlin Heidelberg, 1992.

Rickel, J. W., Intelligent Computer-Aided Instruction: A Survey Organized Around System Components, *IEEE Transactions on Systems, Man and Cybernetics*, 19(1), pp. 40–57 (1989)

Roblyer, M. D., Fundamental problems and principles of designing effective courseware, in *Instructional Design for Microcomputer Courseware*, (D. H. Jonassen, ed.), Lawrence Erlbanm, 1988.

Spector, J. M., and Muraida, D. J., An intelligent Framework for the Creation of Effective Computer-Based Instruction, in *Proceedings of the 34th International ADCIS Conference: Empowering People Through Technology*, Norfolk, VA, November 8–11, 1992, pp. 373–379.

J. SYSTEMS SOFTWARE
1997; 36:233–245

Thimbleby, H., *User Interface Design*, ACM Press, 1990.

Uden, L., CBT the Soft Way, in *Proceedings of the 34th International ADCIS Conference: Empowering People Through Technology*, Norfolk, VA, November 8–11, 1992, pp. 381–400.

Wallsgrove, R., Multi-strategy authoring toolkit for intelligent courseware: better courses for less, in *Learning Technology in the European Communities*, (S. A. Cerri and J. Whiting, eds.), Kluwer Academic Publishers, pp. 573–584, 1992.

Woolf, B. P., Theoretical frontiers in building a machine tutor, in *Artificial Intelligence and Instruction*, (G. P. Kearsley, ed.), Addison-Wesley, pp. 229–267, 1987.

Yazdani, M., and Pollard, D., Multilingual aspects of a multimedia database Of learning materials, in *Learning Technology in the European Communities*, (S. A. Cerri and J. Whiting, eds.), Kluwer Academic Publishers, pp. 495–508, 1992.

Zaharakis, I., Kameas, A., and Pintelas, P., MeT: The Expert Methodology Tutor of GENITOR, The *EUROMICRO Journal: Microprocessing and Microprogramming*, 40, pp. 855–860 (1994).